Egress Gateways with TLS Origination

9 minute read ✔ page test

The TLS Origination for Egress Traffic example shows how to configure Istio to perform TLS origination for traffic to an external service. The Configure an Egress Gateway example shows how to configure Istio to direct egress traffic through a dedicated *egress*

gateway service. This example combines the previous two by describing how to configure an egress gateway to perform TLS origination for traffic to external services.

Before you begin

- Setup Istio by following the instructions in the Installation guide.
- Start the sleep sample which will be used as a test source for external calls.

If you have enabled automatic sidecar injection, do

```
$ kubectl apply -f @samples/sleep/sleep.yaml@
```

otherwise, you have to manually inject the sidecar before deploying the `sleep` application:

```
$ kubectl apply -f <(istioctl kube-inject -f @samples/sleep
/sleep.yaml@)
```

Note that any pod that you can `exec` and `curl` from would do.

- Create a shell variable to hold the name of the source pod for sending requests to external services. If you used the `sleep` sample, run:

```
$ export SOURCE_POD=$(kubectl get pod -l app=sleep -o jsonp
ath={.items..metadata.name})
```

- For macOS users, verify that you are using `openssl` version 1.1 or later:

  ```
  $ openssl version -a | grep OpenSSL
  OpenSSL 1.1.1g  21 Apr 2020
  ```

  If the previous command outputs a version `1.1` or later, as shown, your `openssl` command should work correctly with the instructions in this task. Otherwise, upgrade your `openssl` or try a different implementation of `openssl`, for example on a Linux machine.

- Deploy Istio egress gateway.
- Enable Envoy's access logging

Perform TLS origination with an egress gateway

This section describes how to perform the same TLS origination as in the TLS Origination for Egress Traffic example, only this time using an egress gateway. Note that in this case the TLS origination will be done

by the egress gateway, as opposed to by the sidecar in the previous example.

1. Define a `ServiceEntry` for `edition.cnn.com`:

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: cnn
spec:
  hosts:
  - edition.cnn.com
  ports:
  - number: 80
    name: http
    protocol: HTTP
  - number: 443
    name: https
    protocol: HTTPS
  resolution: DNS
EOF
```

2. Verify that your `ServiceEntry` was applied correctly

by sending a request to http://edition.cnn.com/politics.

```
$ kubectl exec "${SOURCE_POD}" -c sleep -- curl -sSL -o /de
v/null -D - http://edition.cnn.com/politics
HTTP/1.1 301 Moved Permanently
...
location: https://edition.cnn.com/politics
...
```

Your `ServiceEntry` was configured correctly if you see *301 Moved Permanently* in the output.

3. Create an egress `Gateway` for *edition.cnn.com,* port 80, and a destination rule for sidecar requests that will be directed to the egress gateway.

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: istio-egressgateway
spec:
  selector:
    istio: egressgateway
  servers:
  - port:
      number: 80
      name: https-port-for-tls-origination
      protocol: HTTPS
    hosts:
    - edition.cnn.com
    tls:
      mode: ISTIO_MUTUAL
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
```

```
metadata:
  name: egressgateway-for-cnn
spec:
  host: istio-egressgateway.istio-system.svc.cluster.local
  subsets:
  - name: cnn
    trafficPolicy:
      loadBalancer:
        simple: ROUND_ROBIN
      portLevelSettings:
      - port:
          number: 80
        tls:
          mode: ISTIO_MUTUAL
          sni: edition.cnn.com
EOF
```

4. Define a `VirtualService` to direct the traffic through the egress gateway, and a `DestinationRule`

to perform TLS origination for requests to `edition.cnn.com`:

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: direct-cnn-through-egress-gateway
spec:
  hosts:
  - edition.cnn.com
  gateways:
  - istio-egressgateway
  - mesh
  http:
  - match:
    - gateways:
      - mesh
      port: 80
```

```
    route:
    - destination:
        host: istio-egressgateway.istio-system.svc.cluster.
local
        subset: cnn
        port:
          number: 80
      weight: 100
  - match:
    - gateways:
      - istio-egressgateway
      port: 80
    route:
    - destination:
        host: edition.cnn.com
        port:
          number: 443
      weight: 100
---
apiVersion: networking.istio.io/v1alpha3
```

```
kind: DestinationRule
metadata:
  name: originate-tls-for-edition-cnn-com
spec:
  host: edition.cnn.com
  trafficPolicy:
    loadBalancer:
      simple: ROUND_ROBIN
    portLevelSettings:
    - port:
        number: 443
      tls:
        mode: SIMPLE # initiates HTTPS for connections to e
dition.cnn.com
EOF
```

5. Send an HTTP request to http://edition.cnn.com/politics.

```
$ kubectl exec "${SOURCE_POD}" -c sleep -- curl -sSL -o /de
v/null -D - http://edition.cnn.com/politics
HTTP/1.1 200 OK
...
```

The output should be the same as in the TLS Origination for Egress Traffic example, with TLS origination: without the *301 Moved Permanently* message.

6. Check the log of the `istio-egressgateway` pod and you should see a line corresponding to our request. If Istio is deployed in the `istio-system` namespace, the command to print the log is:

```
$ kubectl logs -l istio=egressgateway -c istio-proxy -n ist
io-system | tail
```

You should see a line similar to the following:

```
[2020-06-30T16:17:56.763Z] "GET /politics HTTP/2" 200 - "-"
 "-" 0 1295938 529 89 "10.244.0.171" "curl/7.64.0" "cf76518
d-3209-9ab7-a1d0-e6002728ef5b" "edition.cnn.com" "151.101.1
29.67:443" outbound|443||edition.cnn.com 10.244.0.170:54280
 10.244.0.170:8080 10.244.0.171:35628 - -
```

Cleanup the TLS origination example

Remove the Istio configuration items you created:

```
$ kubectl delete gateway istio-egressgateway
$ kubectl delete serviceentry cnn
$ kubectl delete virtualservice direct-cnn-through-egress-gatewa
y
$ kubectl delete destinationrule originate-tls-for-edition-cnn-c
om
$ kubectl delete destinationrule egressgateway-for-cnn
```

Perform mutual TLS origination with an egress gateway

Similar to the previous section, this section describes how to configure an egress gateway to perform TLS origination for an external service, only this time using a service that requires mutual TLS.

This example is considerably more involved because you need to first:

1. generate client and server certificates
2. deploy an external service that supports the mutual TLS protocol
3. redeploy the egress gateway with the needed mutual TLS certs

Only then can you configure the external traffic to go through the egress gateway which will perform TLS origination.

Generate client and server certificates and keys

For this task you can use your favorite tool to generate certificates and keys. The commands below use `openssl`

1. Create a root certificate and private key to sign

the certificate for your services:

```
$ openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:20
48 -subj '/O=example Inc./CN=example.com' -keyout example.c
om.key -out example.com.crt
```

2. Create a certificate and a private key for `my-nginx.mesh-external.svc.cluster.local`:

```
$ openssl req -out my-nginx.mesh-external.svc.cluster.local
.csr -newkey rsa:2048 -nodes -keyout my-nginx.mesh-external
.svc.cluster.local.key -subj "/CN=my-nginx.mesh-external.sv
c.cluster.local/O=some organization"
$ openssl x509 -req -days 365 -CA example.com.crt -CAkey ex
ample.com.key -set_serial 0 -in my-nginx.mesh-external.svc.
cluster.local.csr -out my-nginx.mesh-external.svc.cluster.l
ocal.crt
```

3. Generate client certificate and private key:

```
$ openssl req -out client.example.com.csr -newkey rsa:2048
-nodes -keyout client.example.com.key -subj "/CN=client.exa
mple.com/O=client organization"
$ openssl x509 -req -days 365 -CA example.com.crt -CAkey ex
ample.com.key -set_serial 1 -in client.example.com.csr -out
 client.example.com.crt
```

Deploy a mutual TLS server

To simulate an actual external service that supports the mutual TLS protocol, deploy an NGINX server in your Kubernetes cluster, but running outside of the

Istio service mesh, i.e., in a namespace without Istio sidecar proxy injection enabled.

1. Create a namespace to represent services outside the Istio mesh, namely `mesh-external`. Note that the sidecar proxy will not be automatically injected into the pods in this namespace since the automatic sidecar injection was not `enabled` on it.

```
$ kubectl create namespace mesh-external
```

2. Create Kubernetes `Secrets` to hold the server's and CA certificates.

```
$ kubectl create -n mesh-external secret tls nginx-server-c
erts --key my-nginx.mesh-external.svc.cluster.local.key --c
ert my-nginx.mesh-external.svc.cluster.local.crt
$ kubectl create -n mesh-external secret generic nginx-ca-c
erts --from-file=example.com.crt
```

3. Create a configuration file for the NGINX server:

```
$ cat <<\EOF > ./nginx.conf
events {
}

http {
  log_format main '$remote_addr - $remote_user [$time_local
]  $status '
  '"$request" $body_bytes_sent "$http_referer" '
  '"$http_user_agent" "$http_x_forwarded_for"';
  access_log /var/log/nginx/access.log main;
  error_log  /var/log/nginx/error.log;
```

```
  server {
    listen 443 ssl;

    root /usr/share/nginx/html;
    index index.html;

    server_name my-nginx.mesh-external.svc.cluster.local;
    ssl_certificate /etc/nginx-server-certs/tls.crt;
    ssl_certificate_key /etc/nginx-server-certs/tls.key;
    ssl_client_certificate /etc/nginx-ca-certs/example.com.
crt;
    ssl_verify_client on;
  }
}
EOF
```

4. Create a Kubernetes ConfigMap to hold the configuration of the NGINX server:

```
$ kubectl create configmap nginx-configmap -n mesh-external
 --from-file=nginx.conf=./nginx.conf
```

5. Deploy the NGINX server:

```
$ kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  namespace: mesh-external
  labels:
    run: my-nginx
spec:
  ports:
  - port: 443
    protocol: TCP
  selector:
    run: my-nginx
```

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
  namespace: mesh-external
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 1
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - name: my-nginx
        image: nginx
        ports:
```

```
    - containerPort: 443
    volumeMounts:
    - name: nginx-config
      mountPath: /etc/nginx
      readOnly: true
    - name: nginx-server-certs
      mountPath: /etc/nginx-server-certs
      readOnly: true
    - name: nginx-ca-certs
      mountPath: /etc/nginx-ca-certs
      readOnly: true
  volumes:
  - name: nginx-config
    configMap:
      name: nginx-configmap
  - name: nginx-server-certs
    secret:
      secretName: nginx-server-certs
  - name: nginx-ca-certs
    secret:
```

```
          secretName: nginx-ca-certs
EOF
```

Configure mutual TLS origination for egress traffic

1. Create Kubernetes Secrets to hold the client's certificates:

```
$ kubectl create secret -n istio-system generic client-cred
ential --from-file=tls.key=client.example.com.key \
  --from-file=tls.crt=client.example.com.crt --from-file=ca
.crt=example.com.crt
```

The secret **must** be created in the same namespace as the egress gateway is deployed in, `istio-system` in this case.

To support integration with various tools, Istio supports a few different Secret formats.

In this example. a single generic Secret with keys `tls.key`, `tls.crt`, and `ca.crt` is used.

2. Create an egress `Gateway` for `my-nginx.mesh-external.svc.cluster.local`, port 443, and

destination rules and virtual services to direct the traffic through the egress gateway and from the egress gateway to the external service.

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: istio-egressgateway
spec:
  selector:
    istio: egressgateway
  servers:
  - port:
      number: 443
      name: https
      protocol: HTTPS
    hosts:
    - my-nginx.mesh-external.svc.cluster.local
```

```
    tls:
      mode: ISTIO_MUTUAL
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: egressgateway-for-nginx
spec:
  host: istio-egressgateway.istio-system.svc.cluster.local
  subsets:
  - name: nginx
    trafficPolicy:
      loadBalancer:
        simple: ROUND_ROBIN
      portLevelSettings:
      - port:
          number: 443
        tls:
          mode: ISTIO_MUTUAL
          sni: my-nginx.mesh-external.svc.cluster.local
```

```
EOF
```

3. Define a `VirtualService` to direct the traffic through the egress gateway:

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: direct-nginx-through-egress-gateway
spec:
  hosts:
  - my-nginx.mesh-external.svc.cluster.local
  gateways:
  - istio-egressgateway
  - mesh
  http:
  - match:
    - gateways:
```

```
      - mesh
      port: 80
    route:
    - destination:
        host: istio-egressgateway.istio-system.svc.cluster.
local
        subset: nginx
        port:
          number: 443
      weight: 100
  - match:
    - gateways:
      - istio-egressgateway
      port: 443
    route:
    - destination:
        host: my-nginx.mesh-external.svc.cluster.local
        port:
          number: 443
      weight: 100
```

```
EOF
```

4. Add a `DestinationRule` to perform mutual TLS origination

```
$ kubectl apply -n istio-system -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: originate-mtls-for-nginx
spec:
  host: my-nginx.mesh-external.svc.cluster.local
  trafficPolicy:
    loadBalancer:
      simple: ROUND_ROBIN
    portLevelSettings:
    - port:
        number: 443
      tls:
        mode: MUTUAL
        credentialName: client-credential # this must match
 the secret created earlier to hold client certs
        sni: my-nginx.mesh-external.svc.cluster.local
EOF
```

5. Send an HTTP request to `http://my-nginx.mesh-external.svc.cluster.local`:

```
$ kubectl exec "$(kubectl get pod -l app=sleep -o jsonpath={.items..metadata.name})" -c sleep -- curl -sS http://my-nginx.mesh-external.svc.cluster.local
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```

6. Check the log of the `istio-egressgateway` pod for a line corresponding to our request. If Istio is deployed in the `istio-system` namespace, the command to print the log is:

```
$ kubectl logs -l istio=egressgateway -n istio-system | gre
p 'my-nginx.mesh-external.svc.cluster.local' | grep HTTP
```

You should see a line similar to the following:

```
[2018-08-19T18:20:40.096Z] "GET / HTTP/1.1" 200 - 0 612 7 5
 "172.30.146.114" "curl/7.35.0" "b942b587-fac2-9756-8ec6-30
3561356204" "my-nginx.mesh-external.svc.cluster.local" "172
.21.72.197:443"
```

Cleanup the mutual TLS origination example

1. Remove created Kubernetes resources:

```
$ kubectl delete secret nginx-server-certs nginx-ca-certs -
n mesh-external
$ kubectl delete secret client-credential istio-egressgatew
ay-certs istio-egressgateway-ca-certs nginx-client-certs ng
inx-ca-certs -n istio-system
$ kubectl delete configmap nginx-configmap -n mesh-external
$ kubectl delete service my-nginx -n mesh-external
$ kubectl delete deployment my-nginx -n mesh-external
$ kubectl delete namespace mesh-external
$ kubectl delete gateway istio-egressgateway
$ kubectl delete virtualservice direct-nginx-through-egress
-gateway
$ kubectl delete destinationrule -n istio-system originate-
mtls-for-nginx
$ kubectl delete destinationrule egressgateway-for-nginx
```

2. Delete the certificates and private keys:

```
$ rm example.com.crt example.com.key my-nginx.mesh-external
.svc.cluster.local.crt my-nginx.mesh-external.svc.cluster.l
ocal.key my-nginx.mesh-external.svc.cluster.local.csr clien
t.example.com.crt client.example.com.csr client.example.com
.key
```

3. Delete the generated configuration files used in this example:

```
$ rm ./nginx.conf
$ rm ./gateway-patch.json
```

Cleanup

Delete the `sleep` service and deployment:

```
$ kubectl delete service sleep
$ kubectl delete deployment sleep
```